"Express Mail" Mailing Label No. **EL917901459US**

PATENT APPLICATION
ATTORNEY DOCKET NO. SUN-P6120-RSH

METHOD AND APPARATUS FOR CLASS INITIALIZATION BARRIERS AND ACCESS TO CLASS VARIABLES IN MULTITASKING VIRTUAL MACHINES

Inventor: Laurent P. Daynes and Grzegorz J. Czajkowski

Related Application

[0001] This application hereby claims priority under 35 U.S.C. §119 to a Provisional Patent Application entitled, "The Design of MVM—a Multitasking Virtual Machine," filed March 15, 2001 by inventors Grzegorz J. Czajkowski and Laurent P. Daynes (Application No. 60/276,049).

[0002] The subject matter of this application is related to the subject matter in a co-pending non-provisional application by the same inventors as the instant application and filed on the same day as the instant application entitled, "Method and Apparatus to Facilitate Sharing Instruction Code in a Multitasking Virtual Machine," having serial number TO BE ASSIGNED, and filing date TO BE ASSIGNED (Attorney Docket No. SUN-P6119-RSH).

BACKGROUND

Field of the Invention

[0003] The present invention relates to computer instruction code. More specifically, the present invention relates to a method and an apparatus for a class initialization barrier in a multitasking virtual machine.

Related Art

[0004] Computer programs written in languages such as JAVA™ are compiled into a platform-independent code, which is executed on a virtual machine, such as a JAVA VIRTUAL MACHINE (JVM). A program that has been compiled into a platform-independent code has the advantage that it can execute on a virtual machine regardless of the underlying central processing unit and native code. The terms JAVA, JVM, and JAVA VIRTUAL MACHINE are trademarks or registered trademarks of SUN Microsystems, Inc. of Palo Alto, California.

[0005] A virtual machine typically includes an interpreter, which interprets the platform-independent code into native code to perform the desired operations. Interpreting the platform-independent code is an inherently slow operation. Therefore, many virtual machine implementations also include a dynamic compiler, which can compile at runtime the platform-independent code into the native code of the machine being used to host the virtual machine. Compiling the platform-independent code into the native code of the host machine can reduce the execution time of the program and, therefore, increase throughput.

[0006] Virtual machines for object-oriented programming languages with dynamic class loading typically load the code of a class when a program resolves a symbolic reference to that class for the first time. The class needs to be initialized

subsequently when the program uses it for the first time. Loading and initialization of a class are two separate events. Initialization of a class may never take place even though the class has been loaded before. In the case of the Java programming language, the initialization of a class consists of executing some code, known as the class's static initializer, that brings the class's variables (also known as the static variables) to a well-defined initialized state. A virtual machine implementation may choose to set a class to the initialized state upon its loading when no action is required to initialize that class. For instance, in the Java programming language, no action is required to initialize a class when this class has no declared static initialization sequence, and either no non-final static variables, or non-final static variables that are all declared to be set to a default value. In this case, a virtual machine implementation can benefits from setting such initialization-less classes to the initialized state upon class loading.

[0007] A class initialization barrier is a sequence of native instructions that calls the virtual machine's runtime to initialize a class if it is not already initialized. Class initialization barriers are included in the implementation of those platform-independent instructions that may result in the very first use of a class (in the case of the Java programming language, there are 4 such instructions: getstatic, putstatic, invokestatic, new). The implementation of a platform-independent instruction can come in two flavors: (i) as a sequence of instruction that is part of the implementation of an interpreter of platform-independent instructions, (ii) or as a sequence of instruction generated by a dynamic compiler of platform-independent instructions.

[0008] Because class initialization barriers need only to be executed once per class, it is common practice in the implementation of non-multitasking virtual machines to have recourse to code-rewriting techniques to remove them and the overhead they induce. In other words, a class initialization barrier can simply be